***2018***



**机器学习 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | MNIST手写数字识别 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计卓1501 |
| 学 号： | U201511086 |
| 姓 名： | 张铭 |
| 指导教师： | 何琨 |

目 录

[1 数据介绍 2](#_Toc965995521)

[1.1 数据背景 2](#_Toc250031909)

[1.2 数据特点 2](#_Toc1364834149)

[1.3 分析目标 3](#_Toc1567070669)

[2 算法设计 4](#_Toc1147034204)

[2.1 总体设计 4](#_Toc722190977)

[2.2 详细设计 5](#_Toc790165197)

[3 实现过程 10](#_Toc72308132)

[3.1 数据加载 10](#_Toc970330618)

[3.2 训练数据 11](#_Toc130339969)

[3.3 分类器验证 11](#_Toc1332774281)

[3.4 预测 12](#_Toc195025620)

[4 结果展示 13](#_Toc1626821613)

[5 总结与心得 18](#_Toc1091020582)

[5.1 课设总结 18](#_Toc1084103683)

[5.2 课设心得 18](#_Toc736699359)

[参考文献 19](#_Toc1508226742)

# 数据介绍

## 数据背景

MNIST手写字体数据集是一个非常经典的用于分类的数据集，包含了60000个训练样本以及10000个测试样本，它是更大的NIST数据集的子集。数据来源真实可靠，常用于机器学习算法研究。

## 数据特点

1. 数据量较大且数据真实，训练数据质量较优；
2. 手写字体的大小已经被规范化；
3. 每张图片大小固定，居中显示字体；
4. 图片为28\*28像素，且每个像素点都为0~255的灰度值。

下图1.1为MNIST数据集的一部分展示。



图 1.1 MNIST部分手写数字展示

## 分析目标

通过训练数据集训练得出分类器，通过此分类器正确识别测试集的手写数字。

# 算法设计

## 总体设计

采用KNN和SVM两种机器学习算法对MNIST手写数据集进行训练和测试。总体上分为四个部分，分别是数据读取，预处理，训练，验证，测试五个部分，各模块之间的关系如下图2.1所示。

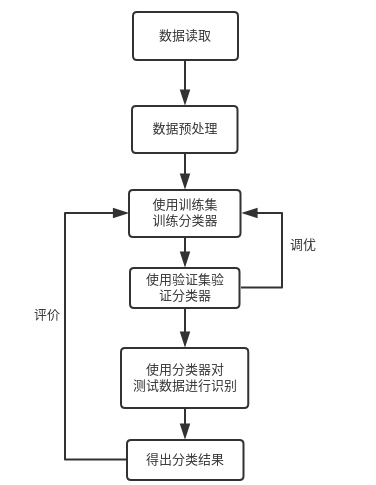


图 2.1 总体模块

1. 使用Python语言完成整个部分；
2. 根据数据集特有的数据结构读取所需要的图片信息，将图片存入自定义的数据结构中；
3. 以此数据结构为基础做数据预处理，将其转换为程序可以处理的数据结构；
4. 从训练数据中选择出10%的数据作为验证集，使用训练数据训练，得出分类器；
5. 使用验证集对分类器进行验证，进一步调整参数；
6. 使用分类器对训练集进行分类，预测出训练集中每个手写字对应的数字；
7. 将预测结果与实际结果相对比，得到预测结果和评价结果。

## 详细设计

此数据集为典型的含有标签的数据集，故采用两种监督学习方法KNN和SVM进行分类。

### 数据集数据结构

从MNIST[1]官方数据集中可以得到如下四个数据集，分别为训练的图片集、标签集，测试的图片集以及测试的标签集，如下表2.1所示。

表 2.1 数据集描述

|  |  |
| --- | --- |
| **数据集** | **描述** |
| train-images.idx3-ubyte | 训练的图片集 |
| train-labels.idx1-ubyte | 训练的标签集 |
| t10k-images.idx3-ubyte | 测试的图片集 |
| t10k-labels.idx1-ubyte | 测试的标签集 |

按照官网中对于数据集的描述，下介绍上表2.1中训练集中的图片和标签的存储数据结构，测试集的图片和标签的存储数据结构同训练集，故不赘述。

1. 标签集数据结构

训练的标签集数据结构描述如下图2.1所示，前4个字节为magic number，值为0x00000801，标识此数据集是否为合法的训练标签集；接下来4个字节为number of items，即数据量，在训练集中共有60000个数据量，故此值为0x0000ea60，接下来为即为每张图片的标签，使用一个字节来表示，如0x0601，即表示两张图片的标签分别是6和1，以此类推。

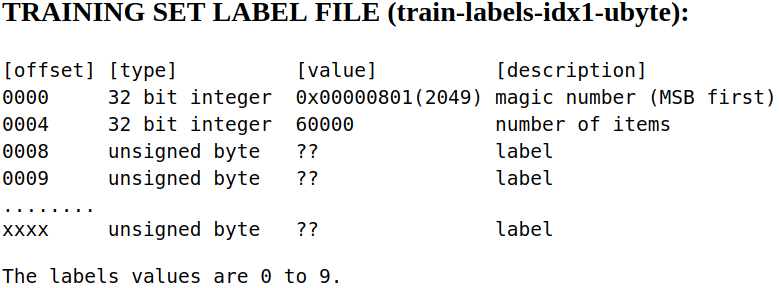


图 2.2 训练集标签数据结构

1. 图片集数据结构

训练的图片集数据结构描述如下图2.3所示，magic number和数据量的结构与标签集的组成方式类似。不同的是在图片像素值之前增加了图片像素行数和列数，为28\*28,故此值为0x0000001c。从第16字节开始为图片的像素值，每个像素值都是0～255范围内，0表示全白，255表示全黑。

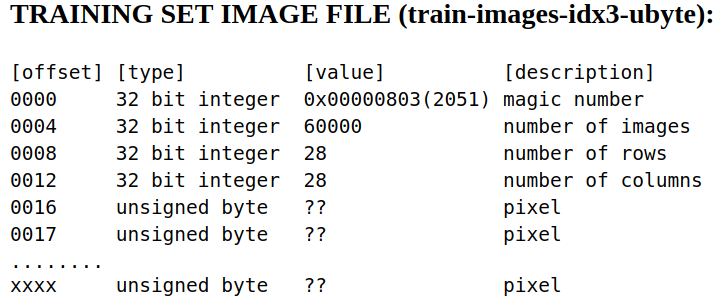


图 2.3 训练集图片数据结构

从Python中读取这两类文件时需要注意文件格式，读取有用的信息。

### 数据预处理

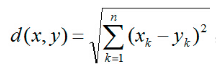
从测试文件和训练文件中读取完标签和图片数据后需要存放在容易处理的数据结构中。读取label后直接存放在一维向量中。将28\*28的图片的像素值展开成1\*784的一维向量，对每个图片均如此处理，得到训练集60000\*784的像素矩阵。

### 训练数据

得到训练集之后，按照10%的比率将训练集分为训练集和验证集，使用90%的训练数据用于训练，分别使用KNN算法和SVM算法进行训练，下简述两种算法的原理。

* KNN（k-NearestNeighbor)）

计算测试图片的像素向量与各训练图片像素向量之间的距离，选择距离最近的K个图片，选择出现次数最多的类别作为测试图片的类别，此处类别共有10种（0～9）。计算向量之间的距离可使用欧式距离公式计算：



例如对如下图2.4数据采用3NN进行计算，则绿色圆被分为红色三角形一类，若按照5NN进行计算，则绿色圆被分为蓝色方块一类。

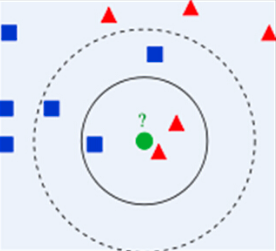


图 2.4 KNN举例

* SVM（Support Vector Machine）

SVM算法多被用于分类，对于线性可分或线性不可分的数据都可以起到不错的分类效果，此处手写字体像素为60000\*784矩阵，是线性不可分的数据，使用多层感知机网络或者SVM分类均可，此处采用SVM进行训练。

SVM是要在训练样本中找到超平面将不同的样本分开，并且对该超平面要求距离两边不同样本点的间距最大。如下图2.5所示，中间实线标明的即为所求的超平面。

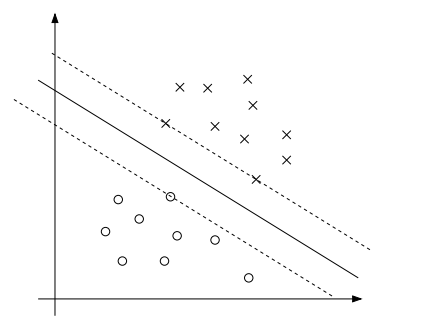
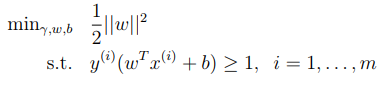


图 2.5 SVM举例

对于该超平面wTx+b=0两边样本点与超平面的最大间距（margin）的求解是SVM分类器至关重要的一点，需要满足如下条件：



根据训练集的图片和标签，使用SVM对训练数据进行分类，再使用SVM对测试数据进行分类。

### 验证数据

使用从训练集中选出的验证集进行分类器的参数调优，并且计算出分类器对于待分类数据的置信度，使用分类器对于验证集计算预测准确度，并且使用误差矩阵（Confusion Matrix）描述误差。误差矩阵可以很明确地表示多少数据被分类正确，多少数据被分类错误。如有150个样本数据，这些数据分成3类，每类50个。分类结束后得到的误差矩阵如下表所示为：

表 2.2 误差矩阵表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **预测** | | |
| 类1 | 类2 | 类3 |
| **实际** | 类1 | 45 | 3 | 2 |
| 类2 | 2 | 45 | 3 |
| 类3 | 1 | 2 | 47 |

说明：每一行之和为50，表示50个样本，第一行说明类1的50个样本有45个分类正确，3个错分为类2，2个错分为类3。使用该误差矩阵可以很容易地看出哪些类别容易被错分。

### 预测数据

使用分类器对手写数字进行预测，并做出预测后的分类结果，与测试用例的标签相比对，得出预测的正确率以及误差矩阵，通过误差矩阵即可查看哪些手写数字是容易被分类器错误分类的。

# 实现过程

## 数据加载

2.2.1节已详细地说明了数据集的结构特点，使用Python的struct和array模块对标签数据和图片数据进行解析，并将图片数据都压缩到一个矩阵中。实现代码如下所示：

def LoadData(image\_path, lable\_path):

with open(lable\_path, 'rb') as file:

magic, size = struct.unpack(">II", file.read(8))

labels = array("B", file.read())

with open(image\_path, 'rb') as file:

magic, size, rows, cols = struct.unpack(">IIII", file.read(16))

image\_data = array("B", file.read())

# append all images to an array

images = []

for i in range(size):

images.append([0] \* rows \* cols)

for i in range(size):

images[i][:] = image\_data[i \* rows \* cols:(i + 1) \* rows \* cols]

return images, labels

使用上述编写的LoadData方法编写加载训练数据的方法loadTrainData，对制定路径的训练标签数据和训练图片数据进行读取，并使用numpy将内置的array类型转换成numpy中的array类型，方便后续的训练。

def loadTrainData(image\_path, label\_path):

image\_train, label\_train = LoadData(image\_path, label\_path)

X = np.array(image\_train)

y = np.array(label\_train)

return X, y

对于测试数据的加载和以上方式相同，只是数据的路径不同。

## 训练数据

* KNN

需要指定邻居数，此处指定K = 5，通过sklearn.neighbors库调用KNeighborsClassifier函数返回KNN的分类器clf，通过clf.fit()函数完成训练过程，训练结束后需要返回clf，以便于对测试集做分类。

def train(train\_image, train\_label):

clf=KNeighborsClassifier(n\_neighbors=5,algorithm='auto',n\_jobs=10)

clf.fit(train\_image, train\_label)

return clf

* SVM

需要指定SVM的核函数和gamma值，为使得样本支持向量数较多，将gamma设置为0.1，并且此处手写数据线性不可分，故使用多项式函数poly作为SVM的核函数。

def train(train\_image, train\_label):

clf = svm.SVC(gamma=0.1, kernel='poly')

clf.fit(train\_image, train\_label)

return clf

## 分类器验证

通过上述已得到的分类器做数据验证，计算出分类器的做分类的置信度confidence，以及验证集的准确度accuracy和误差矩阵confusion matrix，代码如下所示。

def validate(clf, validate\_image, validate\_label):

# calculate confidence, accuracy and confusion-matrix on VALIDATION data

confidence = clf.score(validate\_image, validate\_label)

predicted\_label = clf.predict(validate\_image)

accuracy = accuracy\_score(validate\_label, predicted\_label)

confusionMatrix=confusion\_matrix(validate\_label,predicted\_label)

## 预测

使用分类器对测试集进行预测，并计算出预测准确率accuracy和误差矩阵confusion matrix，代码如下所示。

def predict(clf, test\_image, test\_label):

# calculate accuracy and confusion-matrix on TEST data

predicted\_label = clf.predict(test\_image)

accuracy = accuracy\_score(test\_label, predicted\_label)

confusionMatrixTest=confusion\_matrix(test\_label,predicted\_label)

# 结果展示

* KNN

K = 5，运行程序结果如下图4.1所示。

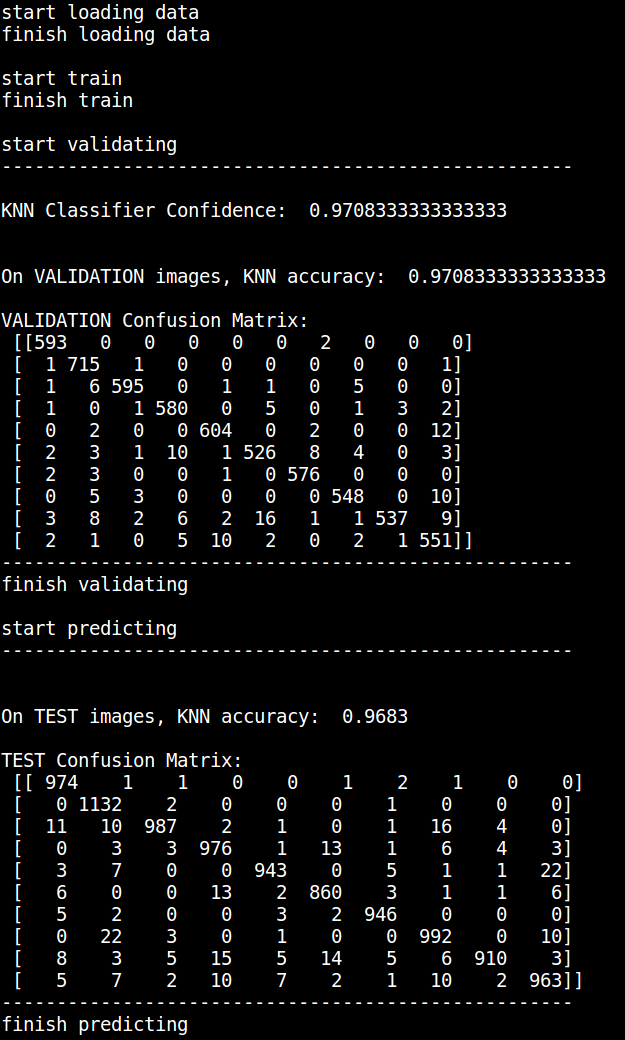


图 4.1 KNN运行结果（K=5）

K=3时分类结果如下图4.2所示。

分类置信度为0.974，分类准确率为0.9685。

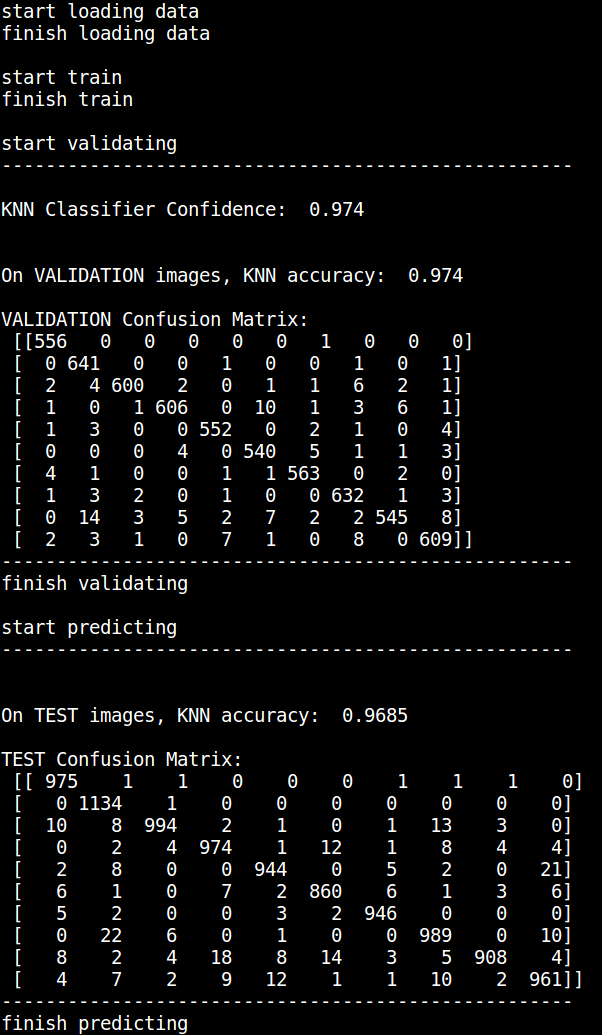


图 4.2 KNN运行结果（K=3）

由上图4.1（K=5）可以得出如下结论：

1. 分类器的置信度为0.9708，分类准确率为0.9683；
2. 从误差矩阵可以看出分类器对数字0、1、6的分类最为准确；
3. 分类器共11次将数字2误分类成了数字0；10次将数字2误分类成了数字3；16次将2分类成了7；13次将数字3分类成了数字5；22次将数字4分类成了数字9；13次将数字5分类成了数字3；22次将数字7分类成了数字2；10次将数字7分类成了数字9；15次将数字8分类成了数字3；14次将数字8分类成了数字5；10次将数字9分类成了3；10次将数字9分类成了7；
4. K为3与K为5时，分类器的分类正确率相差甚微；
5. 分类器容易分类错误的数字：2和0，3和2，3和5，4和9，2和7，7和9，8和3，8和5，9和3。查看数据集可以看出这些数字在书写上确实有很大的相似性，分类器基本达到了预期目标。

* SVM

指定SVM的核函数为poly，gamma值为0.1，分类结果如下图4.2所示，由图可以得出结论：

1. 分类器的置信度为0.981，分类准确率为0.9777；
2. 从误差矩阵可以看出分类器对数字0、1、2、3、6、8均分类较准确；
3. 分类器共9次将数字4误分类成了数字9；共13次将数字5误分类成了数字3；共9次将数字7分类成了数字2；共10次将数字7分类成了数字3；共8次将数字9分类成了数字2，共9次将数字9分类成了数字4；
4. 分类器容易分类错误的数字：3和5，4和9，2和7，7和3，9和2，查看数据集可以看出这些数字在书写上确实有很大的相似性，分类准确率高于KNN算法，分类器达到了预期目标。

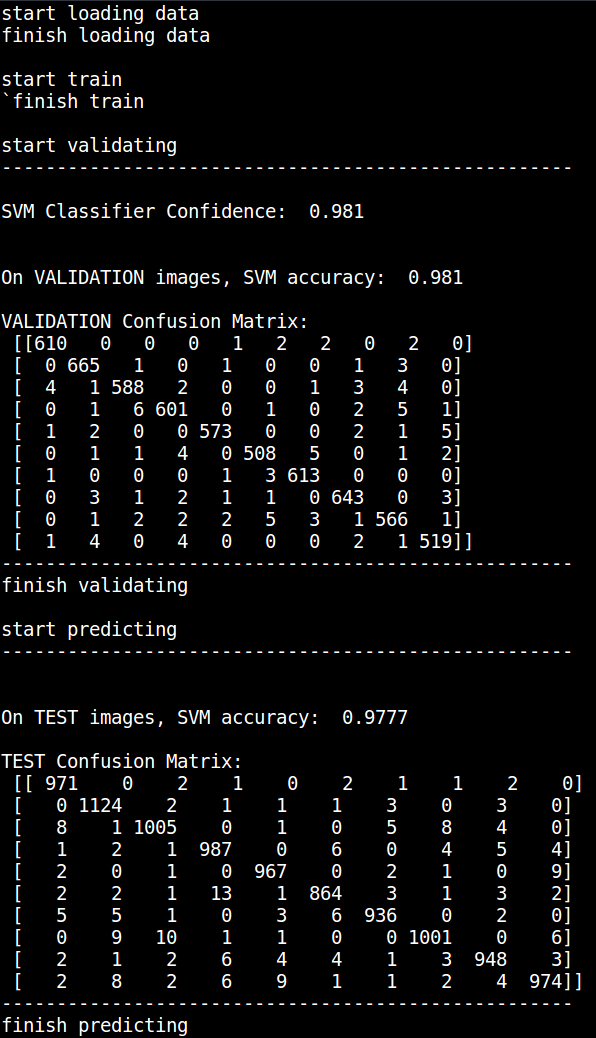


图 4.3 SVM运行结果

# 总结与心得

## 课设总结

本课程设计使用KNN和SVM算法，在经过60000张MNIST手写数字的图片训练下，对10000张测试数据集进行了分类，在较小的错误率下达到了预期的分类效果。

## 课设心得

1. 在具有理论基础之上，通过实践完成了对KNN和SVM算法的使用，体会到了两种算法对于数据分类的效用；
2. KNN和SVM的算法原理虽然简单易懂，但是分类的效果却很好，不愧为机器学习中的经典算法，对具有坚实数学基础的机器学习方向有了更具体的感悟；
3. 以课程设计的方式训练以及完善了对于机器学习算法的理解，有助于更牢固地掌握课堂所学，对于我们学习《机器学习》这门课是非常有益的。

# 参考文献

1. Yann LeCun, Courant Institute, NYU.Corinna Cortes, Google Labs, New York.Christopher J.C. Burges, Microsoft Research, Redmond.THE MNIST DATABASE of handwritten digits [DB/OL]. <http://yann.lecun.com/exdb/mnist/>
2. <https://en.wikipedia.org/wiki/Confusion_matrix>
3. http://scikit-learn.org/